



Better Faster Noise with the GPU

Wyvill, Geoff; Frisvad, Jeppe Revall

Publication date:
2007

Document Version
Publisher's PDF, also known as Version of record

[Link back to DTU Orbit](#)

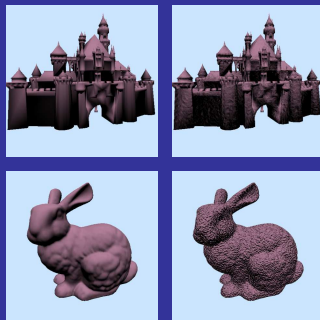
Citation (APA):
Wyvill, G., & Frisvad, J. R. (2007). *Better Faster Noise with the GPU*. Poster session presented at 34th International Conference and Exhibition on Computer Graphics and Interactive Techniques, San Diego, CA, United States.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

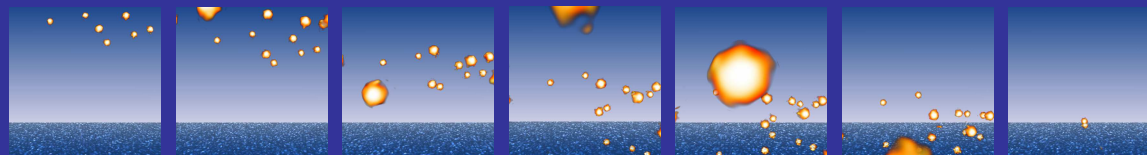
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.



Better Faster Noise with the GPU

Geoff Wyvill
University of Otago
geoff@otago.ac.nz

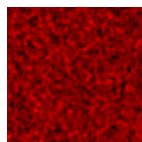
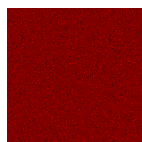
Jeppe Revall Frisvad
Technical University of Denmark
jrf@imm.dtu.dk



Sparse convolution noise [1]

Reference noise

A Gaussian filter over pseudo-randomly placed sources of random value.



Smooth
artifact-free
noise

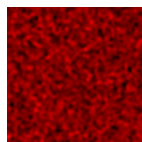
Time:
342.6 sec.

Lossless simplification

Choose a similar filter kernel with compact support.



Gaussian
never zero



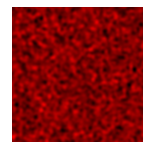
No loss of
quality

Time:
0.70 sec.

GPU implementation

Point rendering

Render each source as a point with width corresponding to kernel size, and use blending. (Similar to spot noise [2].)



Same as before

Time:
0.0056 sec.

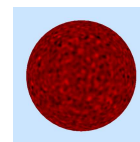
Increasing dimension

Problem: Too many sources (points).

CPU → Find volumes in space where we need sources

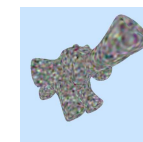
vertex shader → Use depth and perspective to find the size of each point.

fragment shader → Evaluate filter kernel



Solid noise

Time:
0.079 sec.



Elephant slice

Time:
0.052 sec.

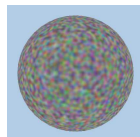
Optimizations

Hints:

- Use a grid.
- Assign an index to each grid cell.
- Use the index to seed pseudo-random numbers for the sources in a cell.
- Render all sources in a cell by sending only cell index to the GPU. Create the sources (points) in a geometry shader.

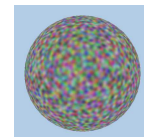
Comparison

Classic Perlin



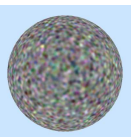
Time: 0.016 sec.
Regularity problems / Texture data needed

Perlin's simplex



Time: 0.011 sec.

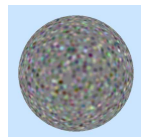
Our noise



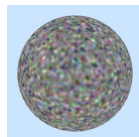
Time: 0.080 sec.
Artifact-free / Computed on the fly

Quality/quantity trade-off

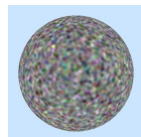
By adjusting the number N of sources per unit volume.



N = 10
Time: 0.030 sec.



N = 20
Time: 0.055 sec.



N = 30
Time: 0.080 sec.

References:

[1] LEWIS, J. P. 1989. Algorithms for solid noise synthesis. In *Computer Graphics (Proceedings of ACM SIGGRAPH '89)*, 23, 3, ACM, 263-270.

[2] VAN WIJK, J. J. 1991. Texture synthesis for data visualization. In *Computer Graphics (Proceedings of ACM SIGGRAPH '91)*, 25, 4, ACM, 309-318.

Timings done on an Centrino 2.13 GHz laptop with a Nvidia Quadro FX Go1400 GPU